UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DESARROLLO DE UNA PLATAFORMA DE TRATAMIENTO Y STREAMING DE VIDEO PARA DIFUSIÓN DE LA CULTURA UTILIZANDO INSTANCIAS DE AZURE

Miguel Porcar Querol
Tutor: Miren Idoia Alarcón Rodríguez

Enero 2018

DESARROLLO DE UNA PLATAFORMA DE TRATAMIENTO Y STREAMING DE VIDEO PARA DIFUSIÓN DE LA CULTURA UTILIZANDO INSTANCIAS DE AZURE

AUTOR: Miguel Porcar Querol
TUTOR: Miren Idoia Alarcón

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2018

Resumen (castellano)

En este Trabajo de Fin de Grado se ha desarrollado la plataforma para un proyecto de emprendimiento llamado Sofastream. En éste se busca satisfacer las necesidades de los artistas de cine independiente para darse a conocer a sí mismos y a sus obras, a través de un sistema de VOD gratuito que se financie con anuncios. De este modo no tienen que asumir más riesgos contratando salas de cine y, por otro lado, los espectadores pueden disfrutar su trabajo remunerándoles a través de la publicidad sin tener que gastar más dinero.

Esta plataforma se ha implementado con tecnologías pioneras como Azure. Con ella, se ha creado un sistema escalable, barato y eficiente con el que se pueden adaptar los gastos según el tráfico y las peticiones existentes. A su vez, Azure cuenta con una red CDN que maximiza la velocidad de transmisión del contenido multimedia a los usuarios finales.

Se busca que Sofastream pueda estar en múltiples dispositivos, por ello, se ha hecho para que funcione en navegadores tanto de ordenadores como de smartphones. Esto facilita la recepción de la plataforma por parte de un mayor número de usuarios, debido a que se puede adaptar a sus necesidades.

Finalmente, se han realizado diversas pruebas con usuarios, ajenos al equipo de desarrollo, resultando su experiencia como exitosa. Se puede concluir que la primera versión está completamente operativa y lista para ser usada.

Palabras clave (castellano)

Emprendimiento, Sofastream, VOD, Azure, CDN, escalable, multiplataforma…

Abstract (English)

At this Bachelor Thesis, a new platform for an entrepreneurship project called Sofastream has been developed. It attempts to satisfy the needs of the independent cinema artists have to get themselves and their artworks known. It is done through a free VOD system financed by advertisement. In this way, they do not have to assume more risks booking movie rooms. On the other hand, the viewers can enjoy their work rewarding it through publicity with no extra money expending.

This platform has been implemented whit pioneer technologies as Azure. With it, a scalable, cheap and efficient system has been created which can adopt itself depending on traffic and existent requests. Also, Azure provides a CDN network which maximizes transmission speed for the multimedia content delivery to the final users.

Sofastream looks forward to being in multiple devices, that is why it has been made to work on multiple devices as computers or smartphones. This eases the reception for a larger number of users, thanks to the fact it can be adapted to their needs.

Finally, several tests have been made with users external to the development team. Their experience has been qualified as successful. It concludes that this very first version is completely operative and ready to be used.

Keywords (inglés)

Entrepreneurship, Sofastream, VOD, Azure, CDN, scalable, multiplatform…

Agradecimientos

A Idoia porque para ella los alumnos no somos sólo números, hay una historia detrás.

A Ricardo Morato por estar conmigo desde el principio. Sin ti no hubiese sido posible.

A mi madre que, aunque elija el camino que le inquieta, siempre está ahí sin importar lo que pase.

A todo el que cree en mí, aunque no haya demostrado nada.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Marco del proyecto

El presente proyecto surge de una idea del estudiante Miguel Porcar a partir de la cual se crea, con ayuda del estudiante Ricardo Morato, una empresa real enmarcada en el sector audiovisual. El propósito principal es la creación de una plataforma que permita ver series, películas y documentales gratis de manera legal y generando beneficios para el dueño de los derechos. El modelo de negocio de esta empresa es el enmarcado como Start-up, es decir, con el desarrollo y el mantenimiento de un sistema se puede dar servicio a una cantidad ilimitada de personas.

Actualmente en España, solamente un 8% de los consumidores de cine pagan por ver una película, en el caso de las series sube a un 12% [1]. Por lo tanto, los productores tienen un gran problema con la monetización de los usuarios que utilizan sistemas de piratería. Es por ello que se decide comenzar un proyecto que solucione esto creando una plataforma para poder visionar este contenido a través de anuncios sin tener que abonarse a una suscripción o hacer un pago puntual.

Por otro lado, se sabe que los artistas que trabajan en el cine independiente tienen grandes barreras para darse a conocer. No disponen de medios para invertir grandes cantidades en marketing ni tienen el suficiente renombre como para poder competir con multinacionales [2], como Sony o Universal. Además, sabemos que una de las principales razones del público para pagar por ver una película es “saber si les va a gustar” antes de ir [3], bien sea conociendo a los actores o al director.

Los costes de las licencias de contenido a nivel industrial son excesivamente elevados cuando no se dispone de inversión inicial, así que se decide focalizar el nicho de mercado al cine independiente. Éste ingresa al año, solamente en España, más de 100 millones de euros únicamente en entradas de cine, contando con las estadísticas anteriores, nos encontramos ante un mercado potencial de más de 1.000 millones de euros [4].

En cuanto a servidores para alojar la plataforma, el club .NET de la Universidad Autónoma de Madrid tiene acceso a cinco cuentas de Azure con 130€ en cada una. Para el desarrollo inicial y ante la escasa demanda, debido a la falta de usuarios, se requerirán solamente dos de ellas.

Miguel Porcar ha realizado todo el trabajo expuesto en este documento, tanto el análisis como el diseño, implementación y las pruebas. Por su parte, Ricardo Morato se ha

encargado de implementar la interfaz de usuario, desarrollo que se menciona muy brevemente en este documento por quedar fuera del alcance de este TFG.

1.2 Motivación

La elección de este proyecto es la consecuencia de un gran número de iteraciones sobre una idea inicial. Todo cambio era motivado porque o la solución aportada no cubría la necesidad de manera apropiada. La mayor parte de las propuestas no tenía nicho de mercado y en consecuencia ni siquiera se planteó un desarrollo de las mismas. Hubo dos proyectos que tras hacer estudios previos sobre su viabilidad y desarrollarlos, los usuarios no terminaron de acoger, ya fuese debido a la complejidad, a las limitaciones de uso o futuros problemas legales.

En primer lugar, se desarrolló un sistema de envío de archivos entre diferentes ordenadores con P2P que podía usarse en el navegador. A su vez, mientras se enviaban, si se trataba de un vídeo o una canción se podían disfrutar en streaming sin tener que esperar la descarga completa. El proyecto fracasó por dos razones: los dos usuarios debían estar conectados a la vez lo que hacía que la velocidad no primase sobre la comodidad que ofrecen servicios como WeTransfer[5]. La segunda razón es que se iba a usar con muy poca frecuencia ya que relativamente pocas veces los amigos van a tener el archivo que se necesita, para ello generalmente se busca en Internet.

El siguiente proyecto fue una red social donde se podían compartir los archivos que uno tenía aprovechando el sistema de envío desarrollado en el sistema inicial. De esta manera se solucionaba el problema de que ambos usuarios tuviesen que estar conectados porque ya habría diferentes personas con ese archivo que lo pondrían a disposición. La intención fue un sistema verdaderamente legal, que respetasen los derechos de autor y la propiedad intelectual. Sin embargo, cuando se empezó a estudiar el comportamiento se vieron los posibles problemas legales en un futuro cercano.

Finalmente, se llegó a la idea de Sofastream: una plataforma de streaming de películas y series gratis. Es muy similar a otras como Netflix o PrimeVideo de Amazon, pero con la particularidad de que no se cobra a los usuarios. El sistema de ingresos es a través de anuncios publicitarios en el visionado, que se repartirían entre los dueños de los derechos de autor y Sofastream. Dicha iniciativa supone una gran novedad entre los servicios de vídeo a demanda actuales, que únicamente ofrecen suscripción. Además, ha suscitado interés y alabanzas entre diversas personalidades del sector del cine.

1.3 Objetivos

El objetivo es crear una plataforma atractiva visualmente y fácil de utilizar, en la que un usuario tenga la capacidad para registrarse, buscar una película y verla, valorarla y compartirla de manera sencilla.

Al no tener acceso a las licencias de contenidos comerciales, se busca contenido de productores independientes que quieren darse a conocer. Por lo tanto, se necesitará implementar un sistema de registro para artistas y otro para que puedan subir el contenido, con el propósito de que posteriormente sea tratado y publicado.

1.4 Organización de la memoria

Esta memoria se divide en 9 secciones:

La sección 2 se describe el estado del arte, es decir, es el análisis de plataformas similares y que son competencia directa en el mercado. En este caso son empresas de streaming de vídeo diferenciadas en el tipo de contenido y en el coste para el usuario.

En la sección 3 se define el proyecto con las herramientas y tecnologías empleadas, así como la metodología de cascada iterativa.

La sección 4 trata el análisis hecho en el que constan los roles de usuario, los casos de uso de éstos y los requisitos funcionales y no funcionales del sistema.

En la sección 5 se describe la arquitectura general del sistema y como ésta se ha tenido que adaptar debido al uso de Azure.

En la sección 6 se define como debe ser la implementación y las particularidades del sistema definido en el análisis y estructurado en el diseño.

La sección 7 contiene las pruebas realizadas tanto por el propio equipo de desarrollo como por usuario ajenos a él y sus resultados.

La sección 8 es la conclusión de esta memoria y las líneas futuras de trabajo.

Finalmente, la última parte del documento contiene un glosario con las palabras clave, las referencias usadas y los anexos.

2 Estado del arte

Se ha llevado a cabo un estudio de plataformas similares, extrayendo de ellas ventajas e inconvenientes, así como puntos fuertes con el fin de conocer esta información antes del análisis y definición de la plataforma que se presenta en este trabajo.

2.1 Plataformas de pago

2.1.1 Netflix

Figura 2-1: Logo Netflix

Conocido por todo el mundo como el estándar del streaming de vídeo a demanda. Netflix[6] posee el mayor número de suscriptores entre todas las plataformas con 104 millones (Julio de 2017)[7] además de uno de los catálogos más extensos tanto de series como de películas. Su plataforma es compatible con smartphones, ordenadores personales y Smart TV de manera que se puede visualizar en cualquier dispositivo adaptándose a él y a su conexión a Internet. De esta manera se mejora la experiencia de usuario, aunque éste no pueda elegir la calidad de la imagen, ya que se elige automáticamente.

2.1.2 HBO

Figura 2-2: Logo HBO

HBO[8] es la gran alternativa y el principal rival de Netflix. Comenzó siendo un canal de televisión, siempre con espíritu innovador, transmitió el primer evento deportivo de pago y acabó desarrollando un sistema de streaming bajo demanda. Su catálogo sigue siendo muy extenso y con grandes éxitos, sin embargo, no cuenta con tantos grandes éxitos en los últimos años como Netlfix.

2.1.3 Amazon Prime Video

Figura 2-3: Logo Prime

Es el servicio de streaming bajo demanda de Amazon [9] creado en el año 2006. De entre los tres gigantes es el que cuenta con un menor número de suscriptores y de contenido. Su inversión en contenido cada vez es mayor pero aún tiene un largo recorrido para alcanzar a HBO y Netflix.

Amazon ha agrupado diferentes servicios (envíos premium, Prime Video, entre otros) en una misma cuenta lo que hace que sea más barato en comparación a Netflix o HBO. Por otro lado, su catálogo dista mucho aún.

2.1.4 Feelmakers

Figura 2-4: Logo Feelmakers

Feelmakers[10] es la plataforma española de streaming bajo demanda que se centra sobre todo en cine independiente, documentales, cortometrajes y animación. Busca una internacionalización hacia países de habla hispana además de España. Los clientes pueden pagar una suscripción mensual o bien pagar por películas o documentales una sola vez. Quizás una medida poco eficaz teniendo en cuenta el la oferta tan pequeña que poseen.

2.2 Plataformas gratuitas

2.2.1 YouTube

Figura 2-5: Logo Youtube

YouTube[11] es la plataforma, adquirida por Google en 2006, que permite a cualquier usuario subir un vídeo y compartirlo. No está orientado a la emisión en producción de películas, documentales o series. Aunque se den casos de ciertos artistas independientes que publican sus obras para darse a conocer, generalmente no da buena imagen publicar una obra en este medio. La difusión en este medio es más sencilla que en otras plataformas debido a la cantidad de gente que lo usa y los recomendadores que tiene.

2.2.2 Flooxer

Figura 2-6: Logo Flooxer

Flooxer[12] es la plataforma creada por Atresmedia. Con ella emiten por un lado el contenido propio, que dividen en vídeos más cortos para llamar la atención del usuario, y por otro lado tienen contenido producido por youtubers o comediantes.

2.2.3 Iqiyi

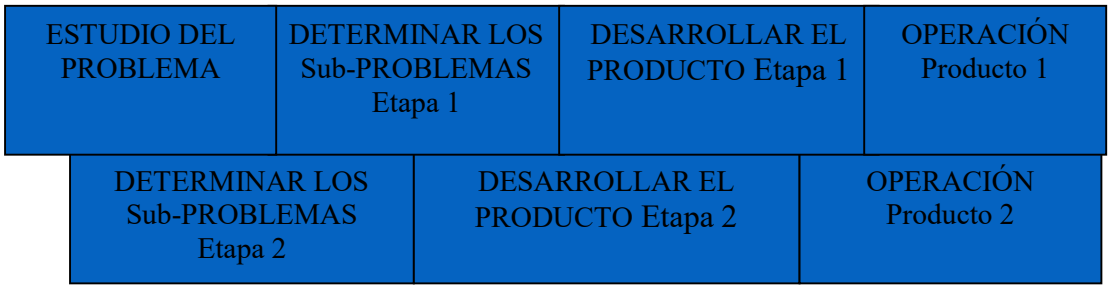
Figura 2-7: Logo Iqiyi

Iqiyi[13] fue fundado principalmente por el buscador Baidu y por el fondo de inversión Providence Equity Partners. Es una plataforma orientada a China con contenido similar al de Netflix o HBO, en la que no se tiene que pagar ningún tipo de suscripción mensual. Además de series y películas, tiene noticiarios, información de finanzas y una tienda integrada. Desafortunadamente sólo opera en China.

3 Definición del proyecto

3.1 Metodología

El modelo de ciclo de vida del software seguido es el incremental iterativo, cuyo esquema se muestra a continuación.



La metodología base utilizada es la mencionada incremental iterativa ya que se han definido, a día de hoy, dos iteraciones con procesos similares obteniendo de cada una de ellas un producto operativo. Por otra parte, se ha combinado un método ágil en el desarrollo que ha sido necesario por el cambio de requisitos que ha sido imprescindible llevar a cabo si se quería conseguir un producto competitivo en el mercado y útil para los usuarios. Los objetivos parciales del producto final han ido modificándose en los primeros inicios del desarrollo de manera ágil hasta conseguir un conjunto de requisitos que sí satisfacen a desarrolladores, usuarios y mercados. No se puede considerar que se ha seguido una metodología ágil pura, entre otras razones, porque el equipo de desarrollo de todo el sistema menos la interfaz de usuario estaba formada únicamente por una persona, el autor de este TFG. Además, el método ágil ha sido utilizado, como se ha mencionado, especialmente en la fase de análisis. Una vez completado el análisis, el desarrollo ha transcurrido por las siguientes fases sin necesidad de realizar cambios continuos. Finalmente, la segunda iteración se basa en el producto obtenido en la primera, lo que encaja plenamente con la metodología incremental iterativa.

La primera iteración está completamente acabada y operativa. En la segunda, el servidor está completo faltando la interfaz por finalizar.

3.2 Tecnologías utilizadas

3.2.1 Plataformas

- Ubuntu 14.04, Windows 10 y Android 5.0-6.0

  Se han usado los tres para asegurarse de la compatibilidad para el cliente en los navegadores tanto de ordenadores como smartphones. Windows 10 se ha utilizado como plataforma de desarrollo para el servidor, ya que será el entorno de ejecución final para el servicio.

- Sublime

Es el editor de texto que se usó por el equipo hasta que se consiguieron licencias para el IDE Visual Studio 2015. Se decidió usar por la ligereza y la rapidez del programa ejecutarse en cualquier entorno, además de un resaltado del código cómodo para la vista.

- Visual Studio 2015

Se trata de un IDE propiedad de Microsoft que al ser miembro del club .NET de la universidad, estuvo a disposición del equipo de Sofastream. Posee un desarrollo con NodeJS bastante intuitivo y además el desarrollo en la nube lo hace instantáneo y sin la necesidad de una configuración exhausta.

3.2.2 Navegadores

- Google Chrome

  Se han usado diferentes versiones para los sistemas operativos de Windows, Ubuntu y Android. Este navegador de Google utiliza el motor V8 de Javascript, que es el mismo que se usa en el entorno NodeJS. Esto hace más sencillo desarrollar en él y después partir a otros navegadores.

- Mozilla Firefox

  Se ha seguido un procedimiento bastante similar al de Google Chrome. Este navegador es el de los creadores del lenguaje Javascript y, en consecuencia, es el que se considera como el estándar.

- Microsoft Edge

  Es el navegador cuyo propietario único es Microsoft y solamente funciona en Windows. Es el que más restricciones tiene por haber implementado la menor cantidad de funcionalidades.

3.2.3 Lenguajes de programación

3.2.3.1 Cliente

- Javascript

  Es el lenguaje usado en cualquier navegador para la programación de scripts que manejen la interfaz de usuario o que se comuniquen en un segundo plano con el servidor.

- HTML5

  Se trata del lenguaje de etiquetas estándar de W3C para estructurar cualquier Web. El uso de la versión 5 es de especial relevancia por la introducción de etiquetas especiales para la reproducción de vídeo de manera nativa. Es una característica clave para el servicio de streaming.

- CSS3

  Es el lenguaje de W3C para modelar una página web, siendo éste el que da la presentación atractiva.

3.2.3.1.1 Librerías

- Angularjs

  Es un framework de Javascript para el desarrollo de la interfaz que sigue el paradigma Model View Controller.

- JQuery

  Se trata de otro framework de Javascript para manejar de manera dinámica elementos de la interfaz del cliente. Mientras que Angular permite un manejo y control de los estados de los elementos, JQuery permite la manipulación de ellos de manera más sencilla que con código nativo.

- Bootstrap

  Es un framework para HTML, CSS y Javascript que permite realizar webs que se adapten automáticamente a la pantalla del cliente, sin importar que sea un ordenador, una Tablet o un smartphone.

3.2.3.2 Servidor

- Javascript

  Se utiliza Javascript en el entorno de NodeJS en la versión 5.5.1. El paradigma de programación es el mismo que en el cliente, sin embargo, el tratamiento de archivos es ligeramente diferente.

3.2.3.2.1 Librerías

- ExpressJS 4.13.4

  Es la librería para manejar las peticiones del protocolo HTTP en NodeJS.

- Body-parser 1.16.1

  Módulo de la librería ExpressJS que permite sacar la información de los formularios HTML.

- Cookie-parser 1.4.1

  Módulo de la librería ExpressJS que facilita la lectura de las cookies en las peticiones de clientes.

- Compression 1.7.0

  Módulo de la librería ExpressJS que, en caso de que el navegador del cliente lo soporte, comprime los archivos al enviarlos.

- Serve-static 1.10.2

  Módulo de la librería ExpressJS que permite configurar una carpeta para servir archivos estáticos de manera óptima.

- Multer 1.3.0

  Librería que permite gestionar los archivos en el servidor, una vez enviados desde un formulario HTML.

- Express-session 1.13.0

  Módulo de la librería ExpressJS que permite gestionar las sesiones de usuario de manera transparente.

- Connect-azuretables 1.0.15

  Librería que permite a Multer guardar los archivos directamente en Azure Storage.

- Node-uuid 1.4.7

  Librería que permite generar números aleatorios.

- Randomstring 1.1.5

  Librería que permite generar cadenas aleatorias.

- Snowmaker 0.5.1

  Librería que se encarga de generar id únicos.
- PassportJS 0.4.0

  Librería que facilita el registro y login de los usuarios.
- Passport-local 1.0.0

  Estrategia de PassportJS para el registro con email.
- Passport-facebook 2.1.0

  Estrategia de PassportJS para el registro con Facebook.
- Passport-twitter 1.0.4

  Estrategia de PassportJS para el registro con Twitter.
- Passport-google-oauth20 1.0.0

  Estrategia de PassportJS para el registro con Google.
- Google-auth-library 0.10.0

  Librería que permite a la estrategia de Google de PassportJS usar su API a través de la autenticación OAuth.
- Googleapis 20.0.1

  Librería que permite el envío de correos con Gmail desde NodeJS.
- Azure-search 0.0.7

  Librería que permite el manejo del servicio Azure Search.
- Azure-storage 0.6.0

  Librería que permite el manejo del servicio Azure Storage.

3.2.4 Instancias de Azure

- Azure Web Services

  Se trata de las instancias dónde se ejecuta el código del backend. Son el servidor propiamente dicho, ya que, toda la lógica sobre la estructura de la plataforma reside en ellas.
- Azure Storage

  Son las instancias que proporcionan el almacenamiento de datos tanto para bases de datos NoSQL como para archivos de cualquier tipo.
- Azure Search

  Estas instancias se utilizan para programar el buscador.
- Azure Media Services

Este tipo de instancias se encargan de codificar y emitir el vídeo en streaming de una manera transparente en cuanto a la escalabilidad.

- Azure Player

  Es el reproductor en el lado del cliente que se acopla a la perfección a los estándares de Azure Media Services.

- Azure CDN

  Azure proporciona una red CDN (Content Delivery Network) para servir el contenido de vídeo en streaming desde puntos optimizados para ello. Son necesarios para la emisión con Azure Media Services.

4 Análisis

4.1 Introducción

En esta sección se describirán los requisitos que debe cumplir la plataforma Sofastream para construir un producto mínimo viable que salga al mercado. A partir de esta primera versión operativa, el sistema sigue en constante desarrollo de nuevas funcionalidades, por lo que, en los casos de uso se plasmarán los requisitos que se hayan previsto necesarios hasta el momento de la redacción de esta memoria.

Con los casos de uso se establecerán las distintas acciones que deberán poder hacer los usuarios, tanto registrados como sin registrar, y los artistas.

Posteriormente, se enumerarán los requisitos funcionales de cada iteración.

4.2 Roles de usuario

Existen tres roles en el uso y desarrollo para esta plataforma:

- Usuario sin registrar
- Usuario registrado
- Artista

A continuación, en los casos de uso se describirán las acciones que podrá realizar cada uno de ellos.

4.3 Casos de uso

Los casos de uso principales de este proyecto se ilustran en los siguientes tres diagramas:

- **Diagrama de casos de uso para los usuarios sin registrar:** estos usuarios no tienen los mismos derechos que uno registrado. Únicamente podrá visualizar y buscar obras, pero nada relacionado con actividades que mejoren su experiencia de uso. Aunque sí contará con la capacidad de registrarse en la plataforma o de iniciar sesión si ya lo está.

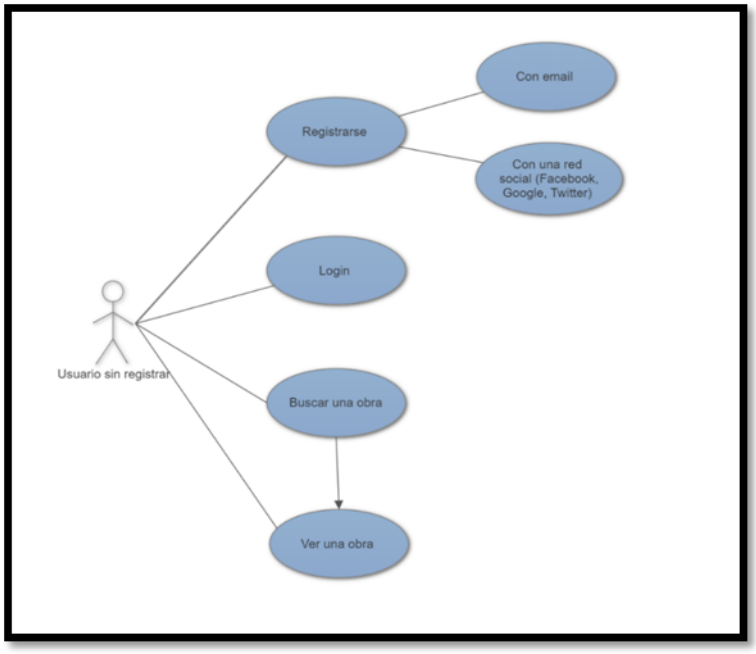


Figura 4-1: Diagrama de casos de uso de usuario sin registrar

- **Diagrama para los casos para los usuarios registrados:** estos usuarios son los verdaderamente interesantes y a los que la plataforma está realmente destinada. Por supuesto, podrán buscar y visualizar una obra, pero además también dispondrán de la posibilidad de suscribirse a las novedades de un artista en concreto y votar o dejar "pendiente" una obra para verla más tarde. En el caso de que alguien tenga material profesional que quiera subir a la plataforma o aparezca en alguna, tendrá la posibilidad de darse de alta como artista.

Figura 4-2: Diagrama de casos de uso de usuario registrado

- **Diagrama para los casos para los artistas:** se trata de un tipo especial de usuarios que disponen de la capacidad de registrar obras y añadir artistas que han participado en ellas. Una vez añadidos todos los participantes, se procederá a notificar a cada uno de sus suscriptores.

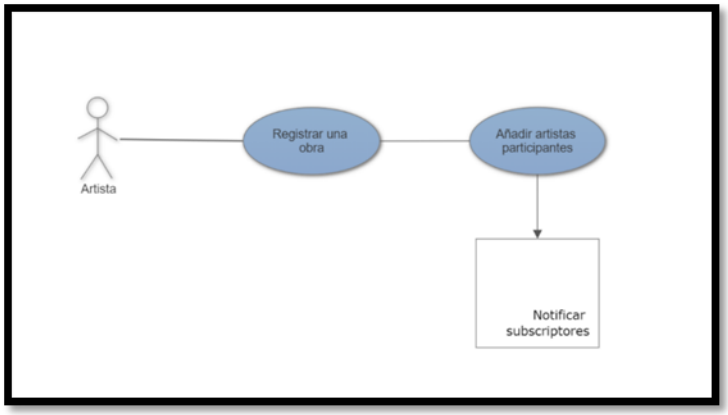


Figura 4-3: Diagrama de casos de uso de usuario artista

4.4 Requisitos funcionales

A continuación, se describirán los requisitos funcionales divididos por iteraciones.

Iteración 1

- **RF-1.- Registro de un nuevo usuario:** un usuario nuevo deberá poder tener la posibilidad de registrarse en el sistema a través de dos vías:
    - **RF-1.1.- Registro con email:** proporcionando un email, no usado previamente, y una contraseña. Se enviará un correo de confirmación con un enlace para ratificar la existencia de esa dirección de correo.

    - **RF-1.2.- Registro a través de las API de Facebook, Google o Twitter:** el usuario accederá con sus credenciales a la red social con la que posteriormente iniciará sesión en el sistema.

- **RF-2.- Login de un usuario:** un usuario ya registrado deberá poder acceder a la plataforma con el mismo método de registro utilizado descrito en el RF-1, es decir, si usa email y contraseña no podrá iniciar sesión a través de ninguna API y viceversa.

- **RF-3.- Registro de un artista:** un usuario registrado podrá darse de alta como artista en el caso de que participe en alguna obra o si tiene los derechos para publicarla. Si el usuario fue registrado con una API de alguna red social, se le pedirá su email. En futuras iteraciones se deberá implementar un sistema para asegurarse de la legitimidad.

    - **RF-3.1- Provisión la información necesaria:** además del comentado email en caso necesario.
        - **ID:** identificador único del artista.
        - **Foto:** foto de perfil del artista.
        - **Nombre artístico.**
    - **RF-3.1- Adición artista al buscador:** Aquí el sistema añadirá su nombre para que pueda ser encontrado a través del buscador especificado en el RF4.

- **RF-4- Creación del buscador**: un usuario podrá buscar tanto artistas como obras a través de un buscador que permita autocompletación y que sea escalable ante muchas búsquedas simultáneas de manera transparente.

- **RF-5.- Registro de una obra:** un usuario artista puede registrar y subir una obra.
    - **RF-5.1.- Provisión de la información necesaria:** ésta es la siguiente:
        - **ID:** identificador único con el que se accederá mediante la url de la forma www.sofastream.com/artwork/{ID}
        - **Título.**
        - **Edad mínima recomendada.**
        - **Géneros:** se seleccionarán 1 ó más géneros de entre una lista, ejemplo: Comedia, Crimen, Drama, etc.
        - **País de origen.**
        - **Sinopsis.**
        - **Fecha de estreno.**
        - **Director.**
        - **Idioma del visionado.**
        - **Poster:** se deberá subir un cartel representativo de la película.
        - **Duración.**
        - **Tipo:** cortometraje, largometraje, documental o serie. Ésta última no se implementará todavía.

    - **RF-5.2- Adición de los participantes de la obra:** un usuario que registra la obra debe proveer los artistas que han participado y su función. Primero buscará en el sistema para ver si están registrados, en caso contrario, procederá a pedir un correo para ponerse en contacto y se creará un artista en el sistema con el email proporcionado. De esta manera se intentará evitar la repetición de artistas en el sistema.

    - **RF-5.3- Almacenamiento y tratamiento de vídeo:** un usuario subirá el vídeo a la plataforma dónde se almacenará y tratará a través de las herramientas convenidas. A su vez, éste se comprimirá y se codificará de manera que el streaming sea adaptable tanto a la conexión de Internet como a la pantalla del usuario.

    - **RF-5.4- Adición de la obra al buscador:** finalmente el sistema añadirá el nombre de la obra al buscador correspondiente al descrito en el RF4.

- **RF-6.- Visualización de una obra:** un usuario podrá acceder a una obra mediante el enlace descrito en el RF-4.1.
    - **RF-6.1- Creación de una página modelo para todas las obras:** se deberá crear una plantilla cuyos campos serán rellenados de manera dinámica según la obra. Los campos a mostrar serán:
        - **Título**
        - **Nombre del director**
        - **Géneros**
        - **Sinopsis**
        - **Actores que participan**
        - **Duración**
        - **País de origen**
        - **Fecha de estreno**
        - **Edad recomendada**

    - **RF-6.2- Customización del reproductor para reproducir la obra:** un usuario lo utilizará para aprovechar la adaptación del streaming desde el lado del cliente de manera transparente para el desarrollador.
- **RF-7.- Búsqueda de una obra u artista:** un usuario podrá, mediante una interfaz gráfica acceder a un buscador común para los artistas y las obras. Se deberán implementar dos tipos de listado:
    - **RF-7.1- Función de autocompletado:** un usuario deberán tener la posibilidad de autocompletar una búsqueda

    - **RF-7.2- Dropdown en el buscador:** cuando un usuario escriba en el buscador, se deberá mostrar un menú al estilo dropdown dónde se muestre una foto y el título o nombre de la obra o del artista, respectivamente**.**

    - **RF-7.3- Página modelo para las búsquedas:** un usuario, tras buscar un nombre, podrá acceder a una página donde se mostrarán todos los resultados que puedan ajustarse.

- **RF-8.- Marcar como "pendiente para ver" una obra:** un usuario registrado podrá guardar obras en algo similar a unos marcadores, dónde las podrá visualizar más tarde.

- **RF-9.- Votación de una obra:** un usuario registrado podrá emitir un voto a través de un sistema de votos para las obras dónde se registre el voto único por usuario, que deberá estar registrado, y se lleve la media. También se deberá registrar las votaciones de cada usuario para crear futuras recomendaciones.

Iteración 2

- **RF-10.- Creación de página para cada artista:** un usuario debe poder acceder a una página modelo estándar que se creará para todo artista que haya aparecido en alguna obra o se haya registrado en el sistema. Se deberá mostrar la siguiente información:
    - **Foto del artista**
    - **Nombre artístico**
    - **Listado de las obras en las que aparece**

- **RF-11.- Suscripción a un artista:** un usuario registrado podrá suscribirse a un artista, de manera que cuando éste aparezca o registre una nueva obra, sea notificado. Este requisito será implementado a través de un botón en la página del artista, definida en el RF8.

- **RF-12.- Notificación a los suscriptores del artista cuando aparece o crea una nueva obra:** como continuación del RF11, se deberá notificar a los suscriptores. La manera para que el usuario se entere se decidirá en el futuro.

4.4.1 Requisitos no funcionales

- **RNF-1.-** La plataforma deberá ser escalable y adaptarse a los aumentos o disminuciones de las peticiones de los usuarios, evitando cortes del sistema por sobrecargas o excesos recursos sin utilizar.
- **RNF-2.-** Los servidores deberán tener costes adaptables a su uso, siendo estos mayores si se necesitan más procesamiento y a la inversa.
- **RNF-3.-** Deberá haber un servicio continuo con una disponibilidad mínima del 99,9%.
- **RNF-4.-** Se tendrán que replicar los datos para protegerse de desastres físicos como corrupción en los discos duros o bien ocurra algún desastre natural.
- **RNF-5.-** Infraestructura suficiente para almacenar y manejar con facilidad cantidades de datos con tamaño mínimo de 5TB.
- **RNF-6.-** El código y la estructura del proyecto deberá ser extensible para que se puedan añadir mejoras continuas.
- **RNF-7.-** Al ser una aplicación Web, habrá que asegurarse que funcione en cualquier navegador tanto de ordenador como de smartphone de cualquier sistema operativo.
- **RNF-8.-** El streaming del vídeo deberá ser adaptable a las diferentes pantallas que puedan tener los usuarios y a la conexión a Internet. No se deben experimentar parones, o lo menos posible.
- **RNF-9.-** Se deberá tener un alto rendimiento por parte del servidor ante muchas peticiones simultáneas y un bajo tiempo de respuesta para cada una de éstas sin que se llegue a consumir excesivos recursos.
- **RNF-10-** Será necesario el uso del protocolo SSH en toda la web para proporcionar seguridad a los usuarios.
- **RNF-11.-** La plataforma en general deberá tener una usabilidad sencilla e intuitiva para el usuario, de manera que, sin conocimientos anteriores ni con necesidad de tutoriales se puedan realizar todas las acciones disponibles en la plataforma.

5 Diseño

5.1 Introducción

En este apartado se va a describir por qué se ha elegido NodeJS como entorno de ejecución y Azure para el hardware, así como sus instancias. Por otro lado, se explicará cual es la arquitectura general del sistema y cómo se ha estructurado la información en las bases de datos y en los buscadores.

5.2 Elección y uso de NodeJS

La elección del entorno de ejecución de NodeJS frente a Java, PHP o Python no es arbitrario. Las características que lo han favorecido han sido:

- Es un entorno monoproceso y monohilo [51], aunque puedan ejecutarse de manera explícita con su librería. Esto evita que por cada petición de un nuevo usuario se genere un nuevo hilo en memoria que consuma más memoria RAM. Además, los cambios de contexto consumen tiempo de procesamiento, al no haber en NodeJS se puede soportar una mayor demanda que con entornos en Java, PHP o Python.
- El tratamiento de peticiones de entrada y salida es no bloqueante [52]. Mientras se lee un fichero se pueden realizar otras acciones, ya que la lectura hace de manera nativa en otro proceso. Con esto se consigue que, aunque se manden grandes archivos a los usuarios o se reciban muchas peticiones, el servicio no quede tan ralentizado como en otros entornos.
- Javascript ya no es un lenguaje interpretado en NodeJS, como si lo ha sido en la mayoría de los navegadores. Ahora éste se compila a lenguaje máquina y se optimiza con varios pasos. En el primero se busca la reorganización del código para que la ejecución sea más rápida. En el segundo, se le dan los tipos más óptimos a las variables, por ejemplo: utilizar un *int* en vez de *double [53]*.

5.2.1 Cómo es la programación con NodeJS

La programación con Javascript es diferente a otros lenguajes a la hora de manejar la concurrencia. Al no haber hilos ni procesos, no se pueden ejecutar diferentes funciones a la vez. A cambio, todas las acciones de entrada/salida se ejecutan de manera asíncrona para evitar bloqueos en el sistema. También hay que tener en cuenta que cuando se ejecuta una

función, esta no se va a interrumpir hasta la operación de entrada/salida y se retomará cuando le toque al evento ejecutarse.

Teniendo en cuenta lo que se acaba de comentar, la programación debe tender a pequeñas funciones con poca carga de trabajo para evitar el bloqueo sobre el hilo de ejecución y se puedan atender más peticiones por segundo, dando una menor sensación de un sistema lento.

5.3 Elección de Azure

De acuerdo con los requisitos no funcionales RNF1, RNF2 y RNF3 comentados en el apartado de Análisis, se necesita estrictamente unos servidores de la *nube*, ya sea Azure, Amazon Web Services (AWS en adelante), Google Cloud, Bluemix de IBM, Arsys, Digital Oceans u otros del estilo.

Partiendo de la oferta de todas las nubes, se buscaron las que mejor se adaptasen a la plataforma definida y se redujo la oferta a Azure, AWS[54] y Google Cloud Computing[55]. De entre estas tres, únicamente Microsoft ofrece planes gratuitos de Azure para estudiantes y para universidades.

Actualmente, Microsoft pone a disposición 4 cuentas de Azure, para los miembros del club .NET de la Universidad Autónoma de Madrid, con 130€ para gastar mensualmente en ellas, de las cuales 2 se utilizan en este proyecto. Se cuenta con el permiso explícito del club siendo además Miguel Porcar quien gestiona el uso equitativo entre todas las personas que soliciten una cuenta.

5.4 Instancias de Azure

Una vez elegido Azure como servicio donde alojar la plataforma, se eligieron las instancias detalladas a continuación para cumplir los requisitos:

- **Azure Web Services:** se trata de la instancia donde se aloja el servicio web. Cumple todos los requisitos, empezando por el RNF1, ya que, escala automáticamente replicando el programa indicado en más o menos instancias según la demanda. También cumple el RNF2, debido a que Azure cobra por minutos usados de cada instancia, por lo que se paga por lo que se usa.

  Se ha usado esta instancia antes que las máquinas virtuales por la facilidad de directamente implementar un servicio únicamente con el código. No hay que

preocuparse de los diversos procesos que pueda haber en la máquina que afecten al servicio ni cómo manejar las direcciones IP al escalar, todo se hace sólo.

- **Azure Storage:** con esta instancia se almacenan todos los datos, tanto para el manejo de las tablas de las bases de datos como para almacenar las obras. Tiene distintas formas de guardado de la información, en esta plataforma se usan las bases de datos no relacionales NoSQL y el almacenamiento de archivos en formato Blob.
  Existe una opción al crear esta instancia que permite definir el tipo de replicado para que sea desde el mismo datacenter de Azure hasta uno global, por lo tanto, también cumple el RNF4.
- **Azure Search:** uno de los inconvenientes de Azure Storage es que una búsqueda por un campo que no se la clave, se consigue en tiempo O(N). Además, tampoco permite la opción de autocompletado ni de expresiones regulares para las cadenas de texto. Es por ello por lo que se usa Azure Search.
  Con esta instancia se pueden indexar los campos que se necesiten para realizar búsquedas eficientes y además permite el autocompletado. Cabe añadir, que el escalado es automático al igual que con las Azure Web Apps.
- **Azure Media Services:** para cumplir con el requisito de tener un streaming adaptable, se necesitan herramientas que compriman el vídeo en diversas resoluciones. Las diferentes herramientas que existen en el mercado para estos propósitos son de pago por licencia mientras que Azure Media Services cobran por uso. Esto permite adaptar nuestros costes al uso.
- **Azure CDN:** se trata de las redes llamadas Content Delivery Network de Azure que permiten servir el vídeo a los usuarios desde puntos más cercanos a ellos en distancia física y con mayor ancho de banda disponible, gracias a que replican las obras en cada uno de ellos.

El RNF3 lo cumplen todas las instancias de Azure, por eso no se ha mencionado anteriormente.

5.5 Arquitectura general

La arquitectura final de cómo se interconectan todas las instancias de Azure entre ellas y como éstas se conectan con los usuarios y artistas es la mostrada en la siguiente imagen:

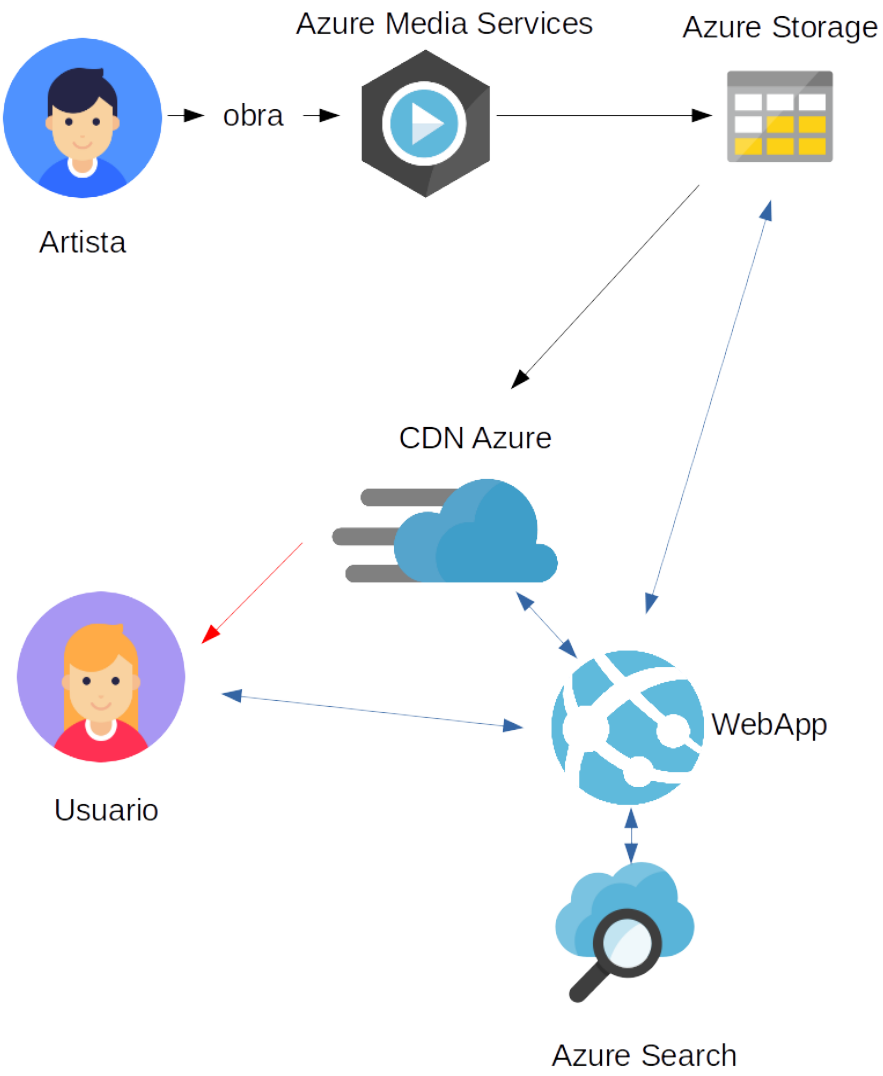


Figura 5-1: Estructura general de Sofastream

La WebApp es el eje neurálgico de la aplicación que se conecta con todas las otras instancias y a su vez las interconecta. Sin embargo, en el esquema no aparece así porque los otros servicios no permiten la comunicación entre ellas a través de eventos con código. Esto hace que las WebApps únicamente sean un puente de unión sin apenas lógica en algunos casos. Así que para facilitar la comprensión del esquema se han suprimido esas conexiones.

El usuario, tanto registrado como no, se comunica principalmente con la WebApp de manera conceptual. Es cierto que las búsquedas las hace directamente al motor de búsqueda de Azure Search porque su API lo permite configurando apropiadamente los permisos, sin embargo, esto sólo ocurre cuando se está ejecutando el buscador con dropdown y además la intención es que el usuario no sepa que lo está haciendo. La

principal búsqueda del sistema es cuando se accede a la página de una búsqueda concreta. En este caso, es la WebApp la que directamente se comunica con Azure Search.

El caso especial de comunicación del usuario es con el CDN de Azure, ya que el usuario no realiza más que una petición a este servicio, únicamente cuando quiere ver la obra.

5.6 Diseño de datos

El diseño de la base de datos de la plataforma se estructura en diversas tablas. En Azure Storage, no hay más claves o indizadores en las tablas que la PartitionKey y la RowKey. La primera sirve para definir las particiones de una tabla para después poder replicarla repartiendo la carga entre cada una de manera óptima. La RowKey es la clave secundaria que puede servir como segundo valor de particionado si es necesario.

La primera es la de usuarios, llamada *Users,* donde se guarda la información relacionada a la actividad de cada uno.

PartKey	RowKey	profilePhoto	tid/gid/fid	username	email
String	String	URL	String	String	String

suscriptions	toSeePublications	viewPublications	voted	artist	artid
Array	Array	Array	Array	Bool	String

Tabla 5-1: Tabla Users

(Misma tabla dividida en dos filas para facilitar la comprensión)

PartKey es el id del usuario y *RowKey* es simplemente un 0. Al no haber una segunda clave necesaria, se sustituye por una constante para mejorar la rapidez en las búsquedas. El campo *tid/gid/fid* en realidad son tres diferentes que se usarán para poner el id de la red social en caso de que el usuario se registrase con una. *ViewPublications* es el campo donde se registran las publicaciones ya visualizadas. Los campos *artist* y *artid* no existen o el primero está a false en caso de que el usuario no se haya registrado como artista.

Para la implementación del login, se han creado dos tablas: una para el registro con correo y otra para el registro mediante redes sociales. Para el registro con correo está la siguiente tabla, llamada *Local*:

PartKey	RowKey	password	userid
String	String	String	String

Tabla 5-2: Tabla Local

En este caso, sucede igual que con la tabla *Users*. No son necesarias ambas *PartKey y RowKey* por lo que solo se usa una de ellas. Para tener homogeneidad con el registro con redes sociales, *PartKey* valdrá siempre la cadena "local" y RowKey será el email usado. *Userid* será el *PartKey* de la tabla *Users.*

Para el registro con redes sociales se usa *Social*:

PartKey	RowKey	accessToken	userid
String	String	String	String

Tabla 5-3: Tabla Social

Sigue la misma estructura que *Local,* donde *PartKey* es el nombre de la red social usada en el registro (Facebook, Google, Twitter) mientras que *RowKey* será el id de ésta. Como particularidad se guarda el accessToken que sirve como clave en el proceso OAuth para asegurarse de la autenticidad del usuario.

La tabla de los artistas, *Artists,* tiene la siguiente estructura:

PartKey	RowKey	email	name	surname	artworks	subscribers
String	String	String	String	String	Array	Array

Tabla 5-4: Tabla Artists

Sigue una organización similar a la tabla *Users*, *PartKey* es el id del autor, que además es la URL de su futura página, y RowKey únicamente es 0. Cabe mencionar que *artworks* es el array de id de publicaciones donde ha aparecido el artista.

Las publicaciones siguen la siguiente tabla:

PartKey	RowKey	actors	age	country
String	String	Array	Integer	String

date	director	gender	lenguage	poster
Date	String	Array	String	URL

streamingUrl	sinopsys	time	title	type
URL	String	Double	String	String

uploader	valoration	votedBy	votes	visualizations
String	Double	Array	Integer	Integer

Tabla 5-5: Tabla Publications

El campo *PartKey* vuelve a ser el id de la publicación y *RowKey* tiene el valor 0. Cabe remarcar el campo *actors* que es un array de los id de los diferentes artistas que han participado. En *votedBy* se almacena un array con tuplas de valor "userid, vote" para registrar los votos de cada usuario y evitar repeticiones. Por otro lado, en *votes* se almacena únicamente el número de votos.

Los archivos, como las fotos de los artistas, los posters de las películas o los vídeos de las obras, se almacenan de una manera diferente. Una de las características de Azure Storage es el almacenamiento de blobs. Éstos se estructuran en *containers*, algo similar a una lista de Blobs. Dentro de ellos, los blobs se diferencian por el nombre y reciben una URL desde la que pueden ser accedidos.

En el container *photos* se guardan todas las imágenes ya sean de posters o de fotos de artistas. No es necesaria hacer una subdivisión, debido a que todos los blobs que hay dentro ya están referenciados y definidos y en las tablas anteriormente explicadas.

Por último, cada vez que se trata un vídeo desde Azure Media Services, se crea un container con todas las diferentes codificaciones de la obra además de un XML que usa el Azure Player para reproducirla.

El buscador sigue una estructura diferente a las tablas anteriores. Debe contener la mínima información necesaria para mostrarla en una búsqueda. La primera tabla que necesitaremos es para el buscador de las obras, llamada "movies", que sigue esta estructura:

url	title	poster
Key, Retrievable	Searchable, Retrievable	Retrievable

Tabla 5-6: Índice de las obras

El campo de la URL es la clave, ya que es el único que no se repite. El título es con el cual se busca y por ello es el campo Searchable. El campo poster únicamente contiene la URL del blob donde se almacena. Todos son *retrievable* (obtenible) debido a que interesa que sean utilizados todos en el lado del cliente para que sean mostrados.

La tabla del buscador de los artistas se llama “artists” y sigue la siguiente estructura:

id	name	photo
Key, Retrievable	Searchable, Retrievable	Retrievable

Tabla 5-7: Índice de los artistas

La semántica es la misma que con “movies”. El id del artista es único, al igual que la URL. Se busca con name y la foto es únicamente la URL donde se almacena la imagen.

6 Implementación

6.1 Creación Azure Web Services

Para la implementación de las Azure Web Services, se selecciona *Web App* desde el panel de Azure. Se introduce una URL provisional y se crea un grupo de recursos para todas las instancias comunes a la plataforma.

La elección del tipo de instancia se hace en la sección de *App Service plan*, como se puede ver en el Anexo A. En primer lugar, se elige la ubicación del servicio, en este caso es *West Europe* para reducir la latencia del servicio para los usuarios de España. Después se elige el plan *S1 Standard* con 1 Core y 1,75GB de RAM que permite escalar hasta 10 instancias de manera automática sin tener que programar nada.

Para la subida y ejecución del código al servidor, se ha decidido que la implementación se haga a través de GitHub. De esta manera, cada vez que se haga un cambio en la rama *master*, se actualizará el servicio reiniciándose automáticamente.

En la sección *Application Settings,* se ha definido la variable *WEBSITE_NODE_DEFAULT_VERSION* a 5.5.1 para que, al ejecutarse la aplicación, la versión del entorno de NodeJS sea esa en concreto. De esta manera, se asegura el mismo funcionamiento desde los ordenadores locales de los desarrolladores y del servidor.

Por último, en el archivo *web.config* se deben declarar diferentes reglas que definan el comportamiento. Para este desarrollo se tomó uno de ejemplo[56] y se añadieron dos reglas de redireccionamiento. La primera, para que siempre se usase https aun cuando se accedía a través del protocolo http. La segunda regla es que en caso de acceder a https://sofastream.com se accediese de manera automática a https://www.sofastream.com.

6.2 Dominio personalizado y SSH

Cuando se crea una instancia de Web App en Azure, se provee de un subdominio de *azurewebsites.net* de manera que al crear *Sofastream* se obtiene *sofastream.azurewebsites.net*. Para conseguir la URL deseada, es decir, www.sofastream.com, se tiene que ir a la Web App creada y acceder a la sección *Custom Domains.* Una vez ahí, existe la opción de comprar un dominio (*Buy Domain)*, donde se introduce el dominio demandado (sofastream.com) y se firma una factura por valor de 11.99$ anuales, como se puede ver en el Anexo B.

El uso de SSH es gratis en el caso de usar el subdominio que ofrecen desde Azure, sin embargo, al usar uno propio se tiene que pagar un certificado personalizado. Para ello se crea una instancia llamada *App Service Certificate*. Posteriormente, se almacena el certificado, se comprueba el dueño del dominio a través de un token de seguridad que se ha obtenido en el paso anterior y finalmente se enlaza con la Web App ya creada.

6.3 Creación de la base de datos

La instancia para el uso de la base de datos y de almacenamiento de las obras se llama Azure Storage. Para crearla, se elige un nombre para la URL (no es pública para el usuario) y una replicación geo-redundante que permita lectura y escritura. La localización para el acceso principal es en *West Europe* para que esté en el mismo datacenter que la Web App principal y disminuya la latencia, como se puede ver en el Anexo C.

Tras haber completado con éxito lo anterior, se guardan las claves y cadenas de conexión de la instancia creada que son usadas para acceder con las API del servicio. Con el fin de evitar problemas en la ejecución, se crean de antemano con un script todas las tablas comentadas en la sección de Diseño y un contenedor de blobs para las imágenes.

Las tablas creadas están vacías, no tienen estructura pues son bases de datos no relacionales y se van creando conforme se van añadiendo tuplas.

6.4 Gestión de peticiones

La librería usada para gestionar los diferentes tipos de peticiones que el servidor puede recibir es ExpressJS. Se pueden declarar todas las rutas que se deseen para cada tipo de petición (GET, PUT, DELETE, etc) y un comportamiento asociado a cada una.

ExpressJS tiene submódulos que se usan en la plataforma para optimizar ciertos procesos. Éstos son: compresión de los archivos en caso de que el navegador del usuario lo permita, una carpeta estática para disminuir la carga al mandar un archivo (.html, .css, imágenes, etc), un parseador de cookies y un parseador de formularios para cuando se envían imágenes adjuntas.

6.5 Registro de usuarios

El registro de usuarios se realiza desde el servidor por motivos de seguridad y simpleza en la implementación. Por ello se utilizará la librería PassportJS que contiene

diferentes módulos que actúan como middleware con las API de Facebook, Google y Twitter además de implementar un módulo para usar correo electrónico y contraseña.

Primero se deben registrar las URL en las que el usuario debe acceder para empezar a registrarse en el sistema. En el caso del registro a través de redes sociales tienen la forma */auth/{{social}}* donde {{social}} es facebook, google o twitter. Para el registro con email es */auth/local,* aunque mientras que las anteriores esperan una petición GET, ésta última espera una PUT con un formulario con campos llamados "email" y "password".

En el caso del registro con las redes sociales, se deben definir otras URL que actúen como callback cuando inicien la sesión en las plataformas de los terceros. Una vez terminado este paso, no se sabe si el usuario estaba iniciando sesión o se estaba registrando. Habrá que hacer una petición en la base de datos y en caso de que el id devuelto por la red social no estuviese usado, se registraría toda la información, mientras que en caso contrario se iniciaría sesión.

PassportJS sí que ofrece una distinción entre registro o inicio de sesión para el caso del email. Aun así, es necesario hacer una comprobación de que el email no esté usado.
La información que se guarda en la tabla de User es la misma sin importar el método de registro, exceptuando los campos fid, gid y tid que guardan el id con el que están registrados en Facebook, Google, Twitter respectivamente. A su vez hay otro campo llamado email en caso de que se haya registrado mediante ese proceso.

Para después permitir el inicio de sesión se guarda la información en las tablas Local o Social según el método usado, acorde a lo definido en la sección de diseño.

6.6 Manejo de sesión

PassportJS y ExpressJS se interconectan con mucha facilidad, pero se necesita implementar dos métodos de serialización y deserialización del usuario de la primera librería que permiten el mantenimiento de una sesión fácilmente. Con la serialización únicamente se guarda el id, mientras que con el otro proceso se carga la información del usuario a través de una petición a la base de datos.

El id del usuario se guarda con una cookie encriptada en el navegador del usuario traduciéndose en el servidor. Desafortunadamente por defecto PassportJS guarda las sesiones en memoria durante la ejecución. Para evitar esto y que se la información esté disponible desde cualquier instancia del servicio se utiliza la librería *connect-azuretables* que la permite almacenar en nuestra cuenta de Azure Storage.

6.7 Login de usuario

El login de un usuario a través de redes sociales es siguiendo el mismo proceso que con el registro salvo que en vez de crear una nueva entrada en la base de datos, ésta se carga. No tiene ninguna complicación a nivel de implementación.

Por otro lado, para implementar el login con el email hay que registrar una nueva URL que recibe una petición POST con los campos "email" y "password". Se comprueba con la tabla Local que coinciden ambos y se registra la sesión con PassportJS.

6.8 Implementación del buscador

Para la implementación del buscador primero hay que crear una instancia de Azure Search desde el portal de Azure. Únicamente hay que indicar la URL que se desea y el tamaño de la instancia, gratuito en este caso debido a la escasez de artistas y obras.
Seguidamente, se crean dos índices: uno para los artistas y otra para las obras. Al primero lo hemos llamado "artists" y al segundo "movies". Se crean desde el panel con las características de cada campo como se indica en la sección de diseño.

Una vez creado cada uno de ellos, se debe configurar el CORS para permitir peticiones desde la URL https://www.sofastream.com. De esta manera las búsquedas desde los navegadores se pueden hacer directamente al buscador sin tener que pasar por los servidores de la plataforma, aliviando la carga de trabajo.

Existen dos tipos de claves de acceso a la instancia del buscador. El primer tipo son del servidor y dan absoluto control sobre la manipulación de los documentos en los índices, no se rigen por ninguna norma CORS. El segundo tipo son las que se usan desde el cliente para hacer las búsquedas. Solamente tienen permitido buscar y se rigen por las normas CORS definidas previamente. Todo esto se puede ver en el Anexo D.

6.9 Registro artista y adición al buscador

En el caso de que un usuario que quiera ser artista se haya registrado a través de una red social, se le pide un email. En caso contrario el proceso sigue. Tras esto, el usuario consulta y reserva un id que no haya sido ya demandado. La reserva se hace guardando el id junto a toda la información en la tabla *UnconfirmedArtists*. En ésta, se deberá guardar, además del id, la información descrita en el requisito funcional 3.1

Posteriormente, se genera una tupla con una cadena aleatoria de 64 caracteres y el email del futuro artista. Esta tupla se almacena en la tabla *ConfirmCode.* Tras esto se

manda un correo a la dirección dada con un mensaje para que se confirme el registro y que es verídico. Este mensaje debe contener un enlace a la URL */confirmArtist/:cadenaAleatoria*. Esta URL es diferente a las otras porque está reservado con un patrón, por lo tanto, lo que se incluya después de la URL */confirmArtist/* quedará almacenado en una variable. El email es enviado a través del servicio de correo Gmail, con la librería no oficial llamada *gmail-node*.

Una vez el artista hace click en el enlace, elimina la tupla creada antes en *ConfirmCode*, se carga la información guardada en *UnconfirmedArtists,* se borra de esta tabla y se guarda en la final que es *Artists*. A su vez, añadimos en la tabla *Users* en el usuario al que está asociado un campo booleano llamado *artist* que vale *true* y otro llamado *artid* con el id de artista creado anteriormente.

Por último, el artista se añade al buscador guardando los valores indicados anteriormente.

6.10 Registro de una obra

En primer lugar, se debe crear una instancia de Azure Media Services para poder gestionar el codificado de los vídeos, así como crear un CDN para poder emitirlas. Las imágenes se pueden ver en el Anexo E.

Se crea una página con un formulario para que el artista pueda dar la información descrita en el requisito funcional 5.1. Para la subida de imágenes se utiliza la librería *multer*. Se configura de manera que cuando se recibe un formulario con una imagen directamente la guarda como en la cuenta de Azure Storage.

En el caso del id de la publicación, se presume que será el título pasado a minúscula y sustituyendo los espacios por "-". En caso de estuviese ya cogida, se añadiría la fecha. Tras esto se guarda la publicación en la tabla *IncompletePublications*.

Una vez hecho lo anterior, se pedirán los artistas participantes en la obra. Primero se dejará que el usuario busque con el buscador y lo seleccione. En caso de no haberse registrado aún se le mandará un email a la cuenta que se proporcione, instándole a que se registre.

El siguiente paso es proporcionar una clave SAS para que el usuario pueda subir el vídeo directamente a la instancia de Azure Storage sin tener que pasar por el servidor. Al subirlo, se tendrá que programar un trabajo a la instancia de Azure Media Services para que comprima y cree un streaming adaptable sobre ese vídeo.

Como Azure Media Services no incluye una API para NodeJS se debe crear un wrapper a través de la REST API para poder manejar el proceso de codificación. Los diferentes pasos son: petición de un AccessToken a través de las credenciales de la instancia, con este token se pide la URL para manejar los trabajos que se encarguen, se crea un "asset" que es el lugar dónde se ha subido anteriormente el vídeo, sobre este "asset" se crea un "locator" que se utiliza para después crear un "job" (el encargo propio que se hace a Media Services).

Un "job" puede durar desde 5 minutos hasta más de 1 hora, por lo que mantenerlo en memoria puede ser peligroso en caso de caída del sistema. Por ello se debe guardar una lista de las tuplas de los trabajos que se estén esperando y su id de publicación en la tabla *PendingJobs*. Cada 10 minutos se va haciendo una petición a Media Services para comprobar si han acabado o no. En el caso de que hayan acabado, se consigue la URL de streaming, se saca la publicación guardada antes en *IncompletePublications,* se borra y finalmente se guarda en *Publications.*

Por último, se añade al buscador para que pueda ser encontrado por los usuarios. Únicamente se guarda en éste el id, el título y la URL del poster, tal y como se ha descrito en el diseño.

6.11 Visualización de una obra

Para la visualización de una obra se necesita una página modelo en la que se visualice toda la información descrita en el requisito funcional 6.1. La información está directamente disponible en la base de datos, por lo que a través de una petición es suficiente.

El reproductor que se usa es Azure Player, que incluye de manera transparente la selección de la calidad con la que se quiere ver el vídeo. A su vez, existe la capacidad de tener un streaming adaptable de manera automática a la calidad de la conexión o la pantalla sin necesidad de codificar nada.

6.12 Búsqueda de una obra o artista

La función de autocompletado, descrita en el requisito funcional 4.1, requiere una implementación en cada uno de los índices de un *Suggester*. Para ello, en la sección de editar o añadir campos a un índice, se selecciona la pestaña Suggester. Se elige el campo

que va a requerir autocompletado, “name” o “title” para “artists” o “movies” respectivamente, y se elige el método “analyzingInfixMatching”.

La librería del cliente de Azure Search envía un JSON en la petición desde el cliente. Ésta deberá añadir la etiqueta “fuzzy: true” cuando se haga una búsqueda. En caso de no crear un *Suggester*, devolverá error.

El dropdown está implementado principalmente en la parte de la interfaz gráfica realizada por Ricardo Morato. Además del estilo, requerirá hacer dos búsquedas consecutivas a los índices creados, ya que, no se permite hacer una simultánea en ambos.

Por último, para la implementación de la página modelo se declara una URL con patrón al igual que se hizo con */confirmArtist/*, en este caso se utiliza */search/:busqueda*. Desde el servidor se hace una búsqueda de la misma forma que se haría desde el cliente y se renderiza la página para que muestre todos los resultados válidos.

6.13 Votación de una obra

En primer lugar, se registra una URL que espera una petición POST en */vote* que contiene un formulario con un voto que es mayor o igual que 0 y menor o igual a 5. Al usuario se le guarda en su tabla *Users*, en el campo *voted,* un JSON en el que cada clave es la película votada y el valor es la valoración dada.

Por otro lado, en la publicación se guarda el id del usuario que ha votado, se suma 1 al número de votos y se recalcula la valoración.

6.14 Implementación sin interfaz gráfica

Todo lo realizado desde este punto no tiene implementación gráfica pero sí desde el lado del servidor.

6.14.1 Marcar obra como pendiente para ver

Se declara la URL que espera una petición POST con patrón */toSeePublication/:id*, donde *:id* es el de la publicación que el usuario desea marcar como “pendiente para ver”. Se carga la lista guardada en el campo *toSeePublications* de la tabla *Users* y se añade el id de la petición, comprobando que no haya repeticiones. Finalmente se guarda de nuevo en la base de datos.

Para realizar el proceso inverso, se reserva otra URL que espera una petición DELETE con patrón */toSeePublication/:id.* En este caso, se procede a eliminar el id de la

lista. Esta petición se realiza cuando el usuario decida eliminarla o bien cuando se visualice la publicación. Se puede entender que darle al play del reproductor cuenta como que se ha visualizado.

6.14.2 Creación de página para artistas

La implementación de la página para artistas se realizará principalmente en la interfaz gráfica. Únicamente se basa en leer la información descrita en el requisito funcional 10 que esté en la base de datos. Además, se deberá incluir un botón que permita a los usuarios suscribirse a ese artista.

6.14.3 Suscripción a un artista

En primer lugar, se declara la URL con patrón */subscribe-artist/:id* que esperará peticiones POST y DELETE para suscribirse y desuscribirse a un artista respectivamente. Se sigue el mismo proceso que con marcar como pendiente para ver más tarde, exceptuando que el campo de la lista en la tabla *Users* es *subscriptions* (y los id son de artistas, no de publicaciones). A su vez, se debe guardar también en el campo *subscribers* de la tabla *Artists* una lista con los id de los usuarios que se suscriben.

6.14.4 Notificación a los suscriptores de un artista

Para la notificación de las novedades de un artista, hay que añadir un paso más al proceso de registro de una obra. Cuando se haya añadido la obra al buscador, se tendrá que añadir a la lista de "pendientes para ver" de cada uno de los suscriptores de cada uno de los artistas.

7 Pruebas

7.1 Pruebas del equipo de desarrollo

Las pruebas realizadas para la comprobación de la correcta implementación son:

- Pruebas unitarias

  Para cada paso en las implementaciones se han llevado a cabo diversas pruebas donde se comprueban el correcto funcionamiento del sistema. Con las herramientas Azure Storage Explorer y Azure Media Services Explorer, se ven reflejados los cambios tanto en las tablas, como en los contenedores de blobs o los *Jobs* de codificación de vídeo. De esta manera, se asegura el correcto funcionamiento de cada implementación.

- Pruebas de integración

  En estas pruebas se integran una a una cada implementación y se comprueba que funcionen correctamente en conjunto. Se trata de la fase más compleja debido a la multitud de combinaciones que se generan entre todos los requisitos.

  Una manera de comprobar la integración del registro ha sido comprobando que el login era correcto. Si además de ello se iba navegando a través de diversas páginas y se ha podido comprobar que usuario visitaba cada página y, de esta manera, se han podido dar por concluida las pruebas de integración de esas implementaciones.

  En cuanto al registro de una obra, se ha comprobado que se mostrase bien en la pantalla principal o en el buscador. Además, una vez se accedía a su página, se mostraba todo tal y como se había insertado. Finalmente, se ha comprobado que se podían visualizar, con lo que se han dado por finalizadas estas pruebas.

  La última prueba ha sido la de suscripción a un artista y la notificación a los usuarios. Al suscriptor del artista le aparece en la lista de "obras pendientes para ver" una obra que se acaba de crear como prueba (y en la que aparecía el artista). Tras esto, se concluía como prueba exitosa.

  El resto de implementaciones no influían entre ellas, por lo que las pruebas de integración no han sido necesarias más allá del correcto funcionamiento general de la plataforma.

- Pruebas de validación

  Estas pruebas se han realizado a través de la interfaz gráfica. Para ello, se ha asegurado que el sistema creado funciona en conjunto, tanto el servidor como la parte gráfica interconectadas.

  Hasta ahora sólo se ha podido comprobar la iteración 1, que es hasta dónde se ha implementado la interfaz de usuario. La prueba ha sido positiva en todos los requisitos funcionales, es decir, del 1 al 9. En cuanto a los requisitos no funcionales, se cumplen todos, aunque no se han podido comprobar el 4 y el 5 porque no ha habido desastres naturales en el caso del primero y no se dispone aún de 5 TB de datos. En ambos casos, Azure asegura que se cumplen los requisitos.

7.2 Evaluación de los usuarios

Para la evaluación del uso de la plataforma, ha participado gente que no forma parte del equipo de desarrollo. El grupo muestral ha sido formado por diversas personas con un nivel alto en ofimática y otro con escasas habilidades en el manejo habitual de los ordenadores. Solamente han podido evaluar la parte implementada con interfaz gráfica, es decir la iteración 1.

Tras estudiar sus valoraciones, se ha concluido que la búsqueda de obras y la visualización de estas es simple y no ocasiona ningún tipo de problemas para ningún tipo de usuario.

Donde sí se han encontrado más dificultades es en el registro. El primer grupo, que posee más habilidades con el ordenador, no ha encontrado problemas. Cabe destacar que el registro con redes sociales era el preferido a usar, siendo el más cómodo y rápido. Sin embargo, el otro grupo de usuarios tenían un comportamiento completamente diferente. Algunos no tenían ni Facebook ni Twitter y no sabían que tener una cuenta de Gmail les valía para registrarse a través de Google. Aunque con el registro con email no encontraron grandes dificultades.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

Este proyecto es la consecuencia de un arduo duro trabajo de emprendimiento. En éste, se ha conseguido pivotar desde una idea inicial insatisfactoria para los usuarios, hasta llegar a una con grandes proyecciones de futuro que ha terminado agradando a todas las personas del mundo del cine que la han conocido. Es un proceso de descubrimiento de necesidades muy exigente, sin apenas guías, que ha requerido muchas horas completar, pero sin duda satisfactorio.

Crear la plataforma Sofastream también ha supuesto un gran aprendizaje en el campo del desarrollo de software. El autor de este TFG ha aprendido a analizar, diseñar, implementar y probar un sistema desde cero. Todos los errores cometidos han acabado como lecciones que en un futuro próximo podrán ser aprovechadas.

8.2 Trabajo futuro

El trabajo futuro que se pretende realizar en Sofastream es:

- Una mejora de la interfaz gráfica diseñada por profesionales del sector. Con ello se conseguirá una plataforma más atractiva y más fácil para vender. La parte que falta por implementar se hará cuando se tenga esto finalmente.
- Implementación de un recomendado para los usuarios que se regirá a través de sus visualizaciones y votos. Con ello se conseguirá una mayor fidelización, ya que se podrá dar a conocer una mayor y mejor oferta del contenido.
- Implementación de un sistema de pago a los productores de contenido de manera que se genere automáticamente. Ahorrará mucho tiempo de trabajo al equipo además de dinero en trabajadores que lo hagan.
- Una ampliación del sector del cine independiente hacia los deportes independientes. Con esto se consigue fomentar diversos deportes que no tienen medios para ser vistos por televisión pero que son practicados por muchos como, por ejemplo: bádminton, rugby, waterpolo, gimnasia rítmica y artística, patinaje sobre hielo, natación sincronizada, etc. Por supuesto incluyendo el deporte femenino dentro de la oferta siempre que sea posible.

Referencias

[1] “Observatorio de la piratería 2016: Resumen ejecutivo”, http://lacoalicion.es/wp-content/uploads/observatorio-pirateria-y-habitos-consumo-2016.-ejecutivo.es_.pdf, accedido: 14/01/18.

[2] “La nueva distribución en España, muchos comensales para tan poco pastel“, https://www.audiovisual451.com/la-nueva-distribucion-en-espana-muchos-comensales-para-tan-poco-pastel/ accedido 14/01/18.

[3] “Guerra de cifras sobre la piratería”, http://cultura.elpais.com/cultura/2016/03/31/actualidad/1459427075_663954.html, accedido 14/01/18.

[4] La fuente pidió explícitamente no ser revelada.

[5] “WeTransfer”, https://wetransfer.com/, accedido: 14/01/18.

[6] “Netflix”, https://www.netflix.com/es/, accedido 14/01/18.

[7] “Netflix supera los 100 millones de suscriptores en todo el mundo”, http://www.eltiempo.com/tecnosfera/novedades-tecnologia/numero-de-suscriptores-de-netflix-en-todo-el-mundo-110256, accedido: 14/01/18.

[8] “HBO”, https://es.hboespana.com/, accedido: 14/01/18.

[9] “Prime Video”, https://www.primevideo.com/, accedido: 14/01/18.

[10] “Feelmakers.com | New ways of viewing”, https://www.feelmakers.com/,

[11] “YouTube”, https://www.youtube.com/, accedido: 14/01/18.

[12] “Flooxer | Los mejores videos de internet. Atresmedia”, http://www.flooxer.com/,

[13] “Iqiyi”, http://www.iqiyi.com/, accedido: 14/01/18.

[14] “Ubuntu 14.04.5 LTS (Trusty Tahr)”, http://releases.ubuntu.com/14.04/,

[15] “Descargar Windows 10”, https://www.microsoft.com/es-es/software-download/windows10, accedido: 14/01/18.

[16] “Android 5.0 (Lollipop)”, https://www.android.com/intl/es_es/versions/lollipop-5-0/, accedido: 14/01/18.

[17] “Sublime Text - A sophisticated text editor for code, markup and prose”, https://www.sublimetext.com/, accedido: 14/01/18.

[18] “Instalar Visual Studio 2015”, https://msdn.microsoft.com/es-es/library/e2h7fzkw.aspx, accedido: 14/01/18.

[19] “Chrome para ordenadores”, https://www.google.com.mx/chrome/browser/desktop/, accedido: 14/01/18.

[20] “Mozilla Firefox”, https://www.mozilla.org/es-ES/firefox/, accedido: 14/01/18.

[21] “Microsoft Edge | Sitio oficial del navegador de Windows 10”, https://www.microsoft.com/es-es/windows/microsoft-edge, accedido: 14/01/18.

[22] “AngularJS – Superheroic Javascript MVW Framework”, https://angularjs.org/,

[23] “jQuery”, https://jquery.com/, accedido: 14/01/18.

[24] “Bootstrap · The most popular HTML, CSS and Javascript library in the world”, https://getbootstrap.com/, accedido: 14/01/18.

[25] “NodeJS”, https://nodejs.org/dist/v5.12.0/, accedido: 14/01/18.

[26] “Express – Infraestructura de aplicaciones web Node.js”, http://expressjs.com/es/, accedido: 14/01/18.

[27] “BodyParser”, https://github.com/expressjs/body-parser, accedido: 14/01/18.

[28] “cookie-parser”, https://www.npmjs.com/package/cookie-parser, accedido: 14/01/18.

[29] “compression”, https://github.com/expressjs/compression, accedido: 14/01/18.

[30] “serve-static”, http://expressjs.com/en/resources/middleware/serve-static.html, accedido: 14/01/18.

[31] “Multer”, https://github.com/expressjs/multer, accedido: 14/01/18.

[32] “express-session”, https://github.com/expressjs/session, accedido: 14/01/18.

[33] “connect-azuretables”, https://www.npmjs.com/package/connect-azuretables, accedido: 14/01/18.

[34] “uuid”, https://www.npmjs.com/package/uuid, accedido: 14/01/18.

[35] “randomstring”, https://www.npmjs.com/package/randomstring, accedido: 14/01/18.

[36] “snowmaker”, https://www.npmjs.com/package/snowmaker, accedido: 14/01/18.

[37] “Passport”, http://www.passportjs.org/, accedido: 14/01/18.

[38] “passport-local”, https://github.com/jaredhanson/passport-local, accedido: 14/01/18.

[39] “passport-facebook”, https://github.com/jaredhanson/passport-facebook, accedido: 14/01/18.

[40] “passport-twitter”, https://github.com/jaredhanson/passport-twitter, accedido: 14/01/18.

[41] “passport-google-oauth20”, https://github.com/jaredhanson/passport-google-oauth2, accedido: 14/01/18.

[42] “Google APIs Node.js Client”, https://github.com/google/google-api-nodejs-client, accedido: 14/01/18.

[43] “node-azure-search”, https://github.com/azure-contrib/node-azure-search, accedido: 14/01/18.

[44] “Microsoft Azure Storage SDK for Node.js”, https://github.com/Azure/azure-storage-node, accedido: 14/01/18.

[45] “Creación de una aplicación web de Node.js en Azure”, https://docs.microsoft.com/es-es/azure/app-service/app-service-web-get-started-nodejs, accedido: 14/01/18.

[46] “Documentación de Azure Storage”, https://docs.microsoft.com/es-es/azure/storage/, accedido: 14/01/18.

[47] “Documentación de Azure Search”, https://docs.microsoft.com/es-es/azure/search/, accedido: 14/01/18.

[48] “Documentación de Azure Media Services”, https://docs.microsoft.com/es-es/azure/media-services/, accedido: 14/01/18.

[49] “Azure Media Player”, http://amp.azure.net/libs/amp/latest/docs/index.html, accedido: 14/01/18.

[50] “Documentación de CDN”, https://docs.microsoft.com/es-es/azure/cdn/, accedido: 14/01/18.

[51] “Overview of Blocking vs Non-Blocking”, https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/, accedido: 14/01/18.

[52] “Asincronicidad en Node.js”, http://nnombela.com/blog/2012/03/21/asincronicidad-en-node-dot-js/, accedido: 14/01/18.

[53] “Is Node.js compiled or interpreted language?”, https://hashnode.com/post/is-nodejs-compiled-or-interpreted-language-cijylh0ed00keco5318e1em8p, accedido: 14/01/18.

[54] “AWS, Cloud Computing”, https://aws.amazon.com/es/, accedido: 14/01/18.

[55] “Google Cloud Computing”, https://cloud.google.com/?hl=es, accedido: 14/01/18.

[56] “Sample of code”, https://gist.github.com/davidebbo/8ad0d30ac1b1aa3d0334, accedido: 14/01/18.

Glosario

API	Application Programming Interface.
Backend	Parte del software donde se procesa la entrada del Frontend.
CDN	Content Delivery Network.
CORS	Cross-Origin Resource Sharing.
Instancia	Capa de abstracción sobre un tipo de servidor de la nube.
Login	Proceso de inicio de sesión de un usuario.
Middleware	Software que proporciona un enlace entre aplicaciones de software independientes.
Montetización	Acción de rentabilizar un producto o servicio.
NoSQL	Sistema de gestión de datos que difiere del modelo SGBDR tradicional.
P2P	Peer-to-Peer.
Start-up	Empresa innovadora que generalmente utiliza la tecnología como pieza clave en su modelo de negocio.
Smart TV	Televisión con integración de internet.
VOD	Video on demand.
Youtuber	Persona cuya profesión es subir videos a YouTube de diversa índole.

Anexos

A Azure Web App

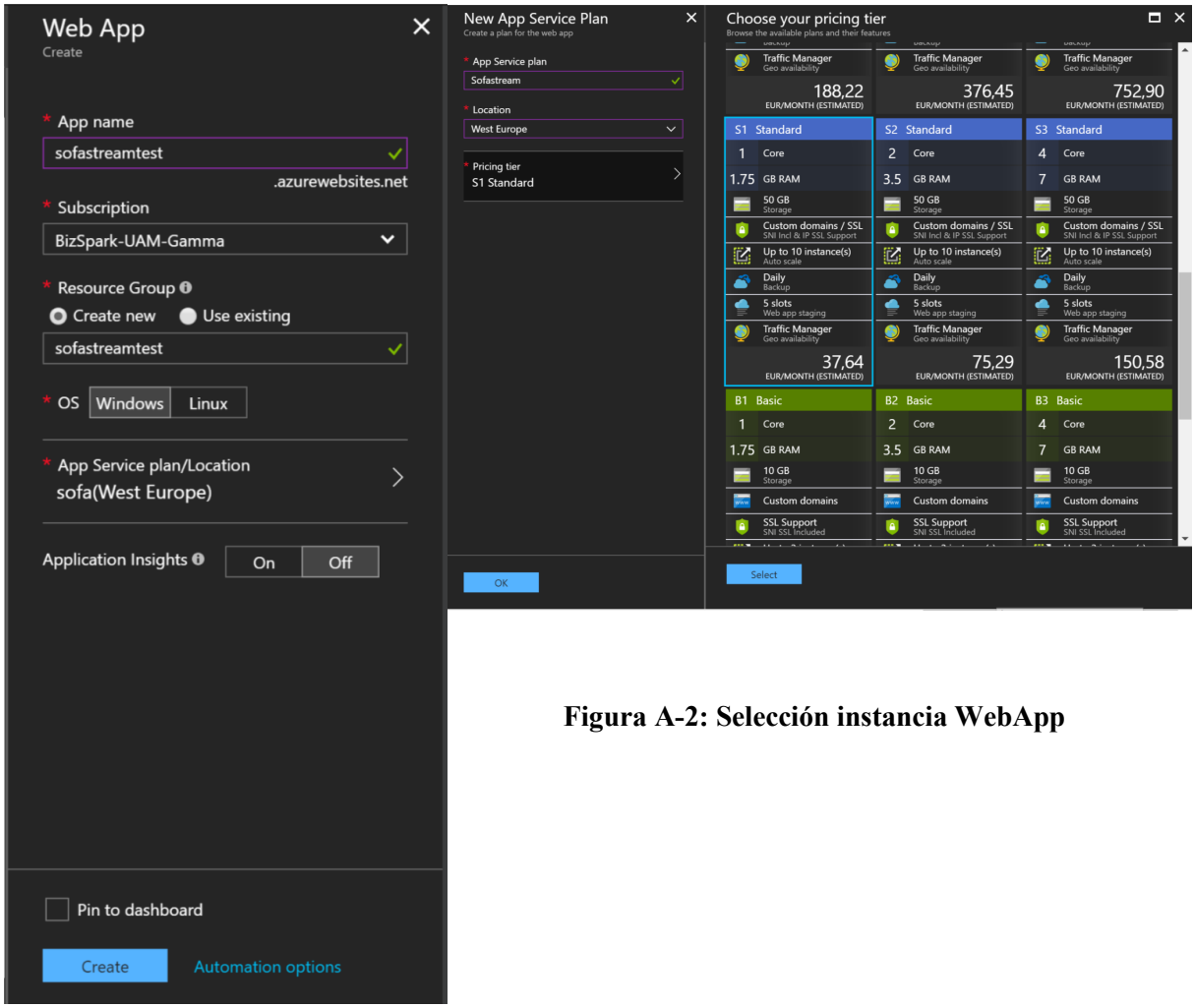


Figura A-2: Selección instancia WebApp

Figura A-1: Instanciación WebApp

B Custom Domain

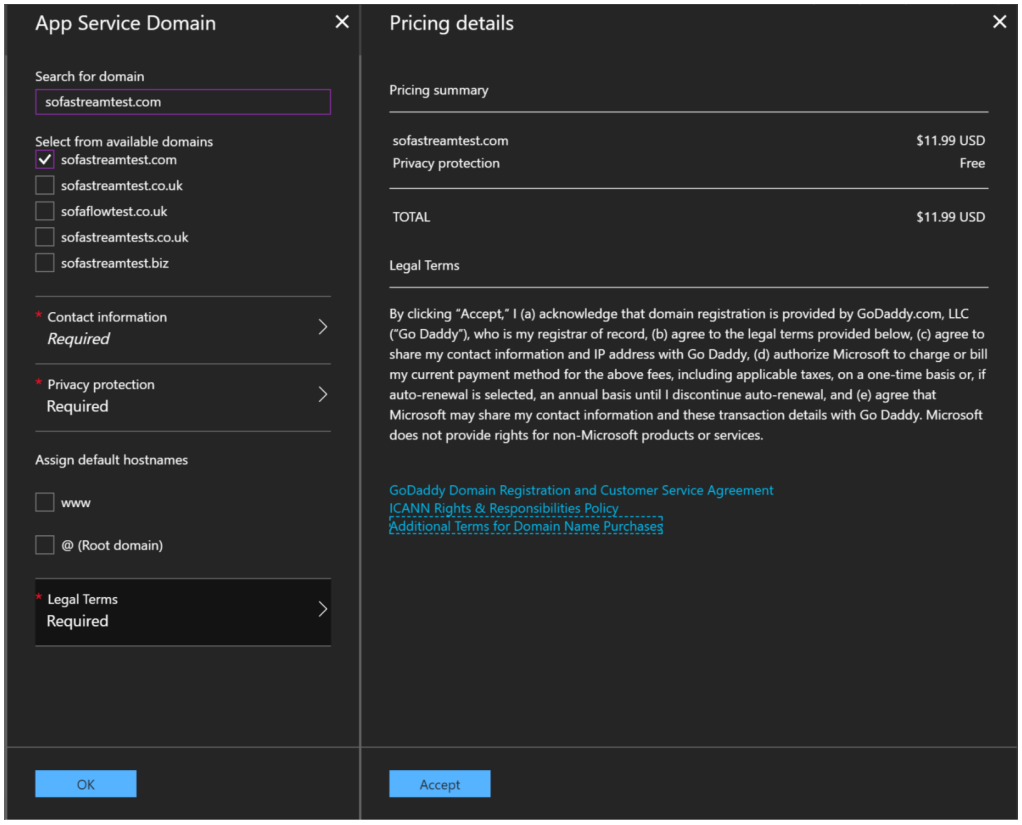


Figura B-1: Creación dominio

C Azure Storage

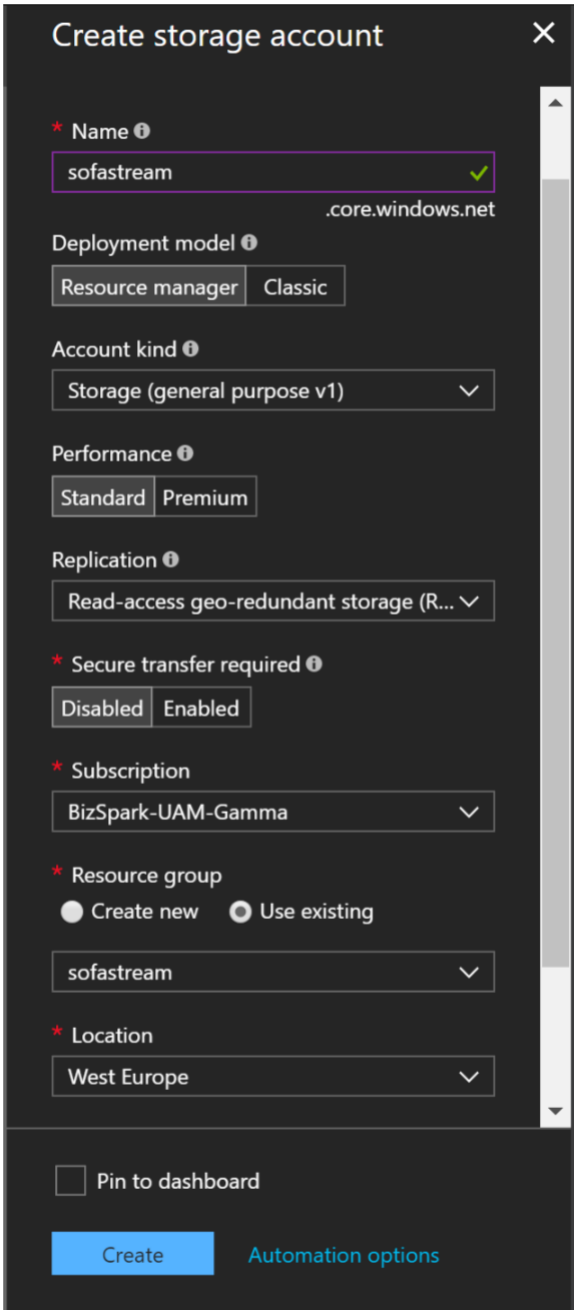


Figura C-1: Creación Storage

D Azure Search

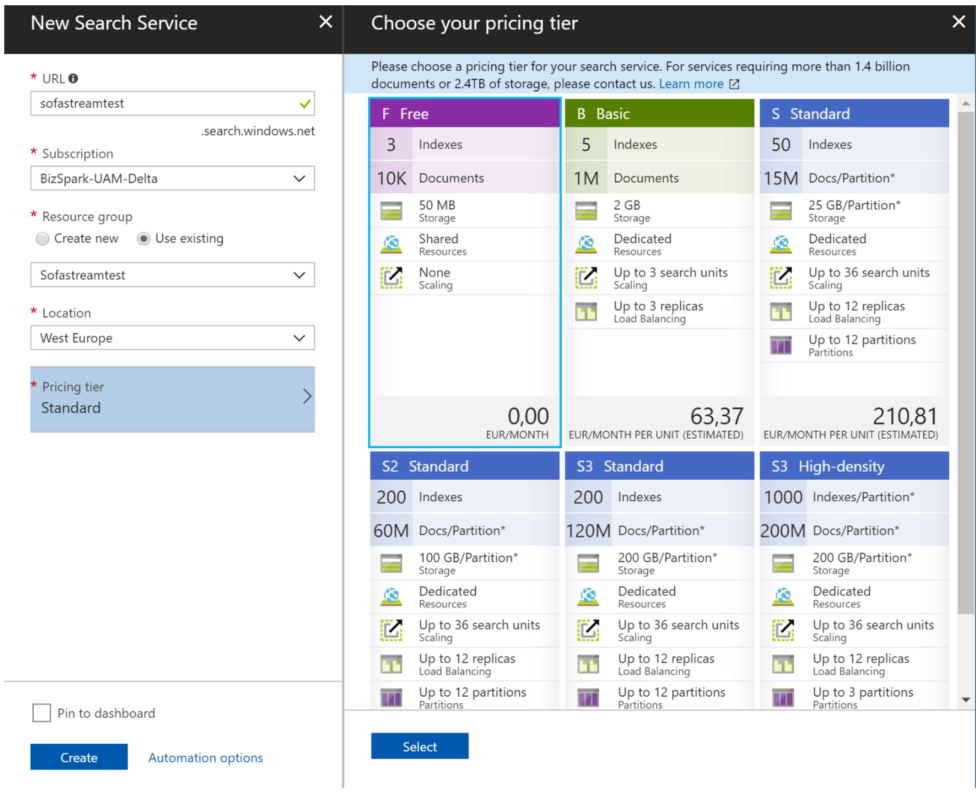


Figura D-1: Creación buscador

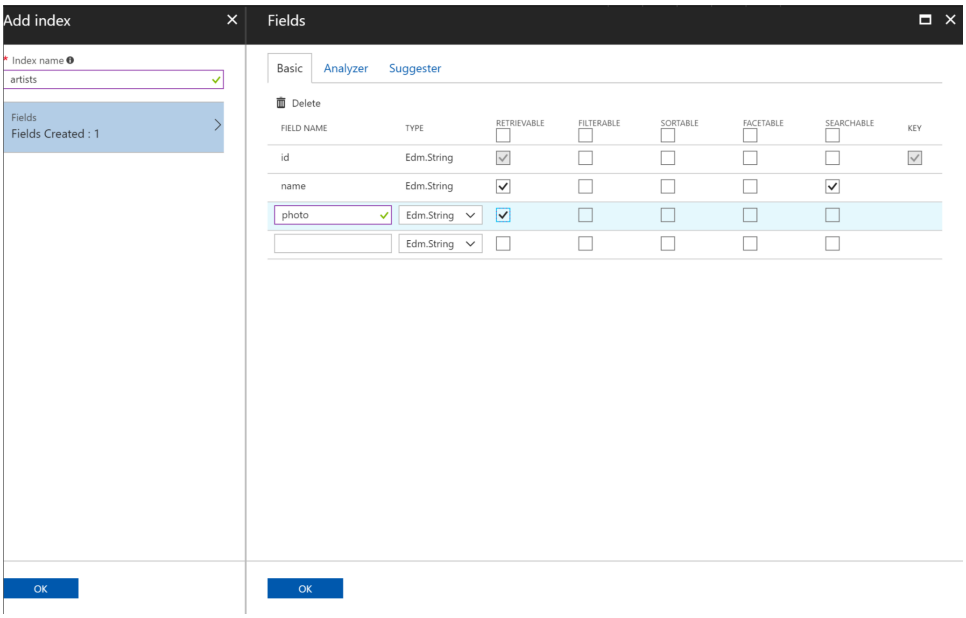


Figura D-2: Creación índice

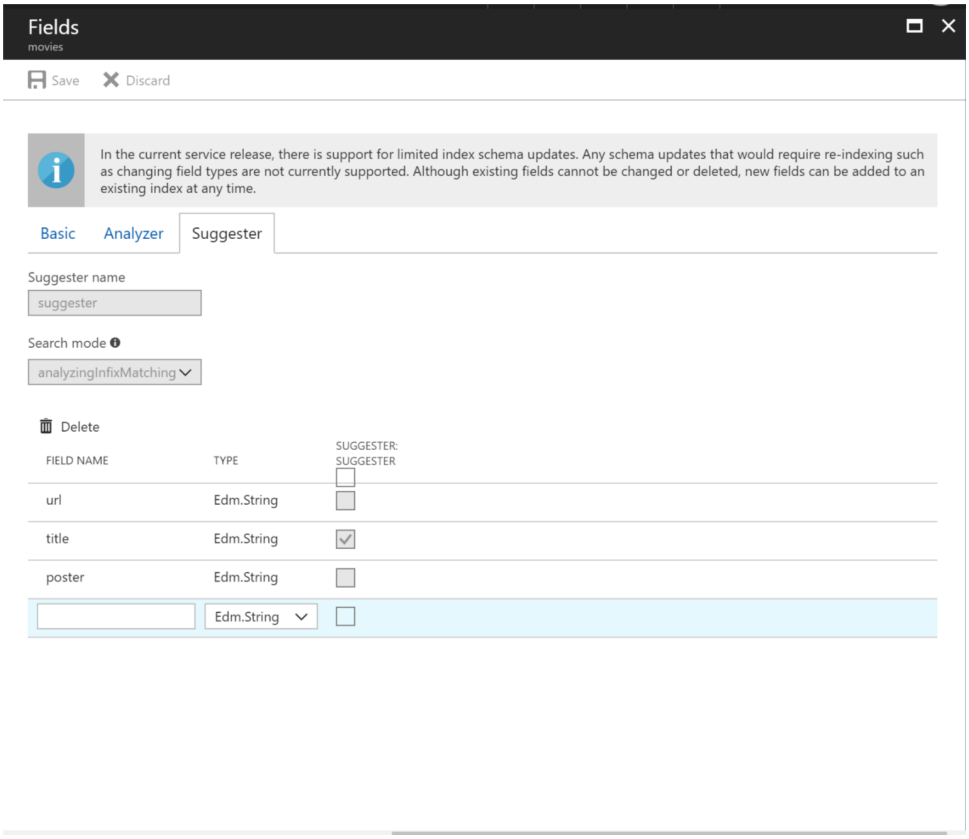


Figura D-3: Configuración Suggester

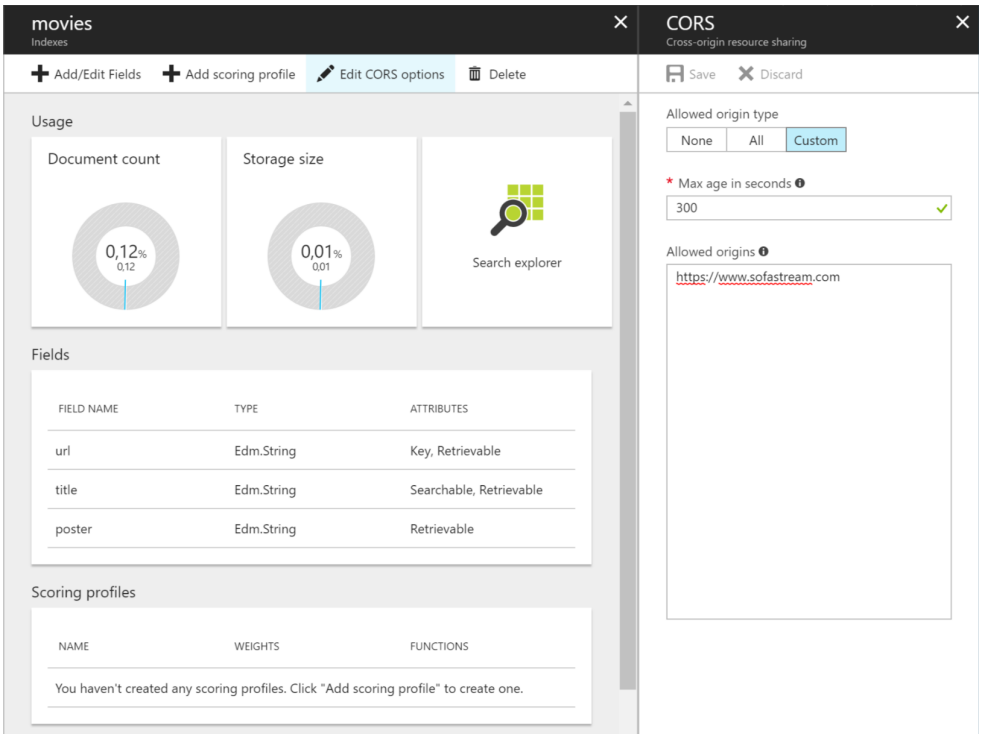


Figura D-4: Configuración CORS

E Azure Media Services

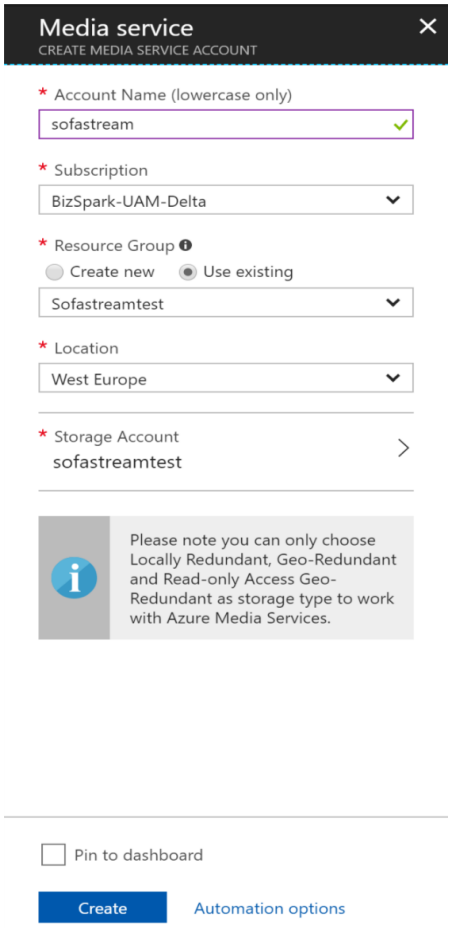


Figura E-1: Creación Media Services

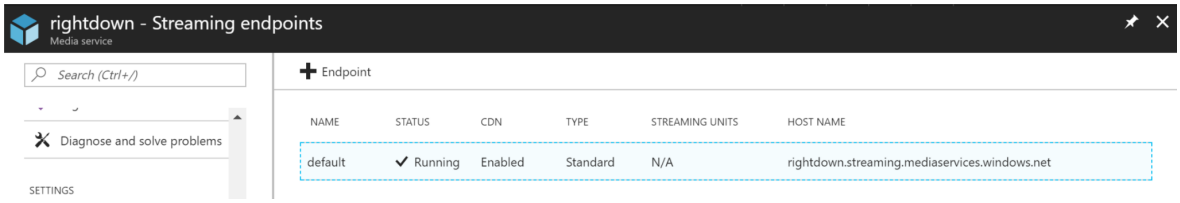


Figura E-2: Configuración CDN endpoints